Team Jefferson

DARPA Urban Challenge 2007

Technical Paper

June 1, 2007

Author
Paul J. Perrone
CEO
Perrone Robotics
pperrone@perronerobotics.com
P.O. Box 4698; Charlottesville, Virginia 22905
(434) 823-2833 (Work) (434) 823-2296 (Fax)

Executive Summary

Team Jefferson's approach has been to leverage a highly configurable and programmable general purpose robotics software platform to enable the rapid and low-cost development of their autonomous ground vehicle "Tommy Jr" for the DARPA Urban Challenge. The team's approach leverages the MAX robotics software platform from Perrone Robotics to facilitate the rapid integration of after-market hardware dropped into an existing commercial vehicle platform for a total cost of approximately $50,000 in parts. In just a few hours of time, the MAX software platform was configured to yield a new vehicle fully capable of autonomous navigation and obstacle avoidance. An additional two man-months of time were spent readying the vehicle with new rules dropped in and configured to meet specific Urban Challenge navigation and traffic evaluation requirements. Tommy Jr's sensor and processing intelligence was implemented using MAX on two low-power and low-cost processing cards costing less than $200 a piece. This paper describes the integrated automotive, mechanical, sensory, hardware, and software architecture that has imbued Tommy Jr with its ability to navigate the DARPA Urban Challenge.

1.0 Introduction

Team Jefferson was founded by Perrone Robotics, Inc. (PRI) in 2004 as a means to demonstrate how its general purpose robotics platform, trademarked MAX, and commercial extensions to MAX could be used to rapidly and affordably build and integrate highly complex robotics applications such as those considered by the DARPA Grand Challenge. Team Jefferson advanced as a semi-finalist in the 2005 DARPA Grand Challenge with its off road desert racing vehicle 'Tommy'. Team Jefferson reformed to compete in the 2007 DARPA Urban Challenge and consists of software engineering and management personnel from Perrone Robotics, mechanical and electrical engineering students and professors from the University of Virginia, a core group of industry luminaries and advisors, and their new commercial purpose city racing vehicle 'Tommy Jr.'. As of June 2007, Team Jefferson's core sponsors for the DARPA Urban Challenge included Perrone Robotics, the University of Virginia, Sun Microsystems, BD Metrics, 45Fix.com, TDFund, SICK, and Assured Technologies.

2.0 Overview of Approach

The problem of the DARPA Urban Challenge is defined by a series of factors by which the autonomous vehicles competing will be evaluated [1, 2]. These factors include: 1) *basic navigation* of a series of checkpoints with a route planned over a route network while staying in vehicle lanes, obeying speed limits, avoiding collisions, maintaining safe distances from other vehicles, safely passing vehicles, and obeying other basic navigation rules of the road; 2) *basic traffic* rule following for obeying precedence order at intersections and maintain safe following distances; 3) *advanced navigation* of obstacles in open zones and parking lots as well as navigating under the condition of contingencies such as roadblocks, GPS outages, and poorly defined lanes; 4) *advanced traffic* rule following for vehicle merges and left turns across traffic as well as implementing safe traffic navigation rules under contingencies such as defensive driving, traffic jams, and traffic in open zones. Implicit in all of these problems to address is also the underlying problem of having a vehicle platform with the sensors to detect various physical world conditions to trigger rules of behavior, actuators to control vehicle speed and direction

based on desired vehicle position and orientation, and the computational capabilities to implement the desired rules and behaviors.

The approach of Team Jefferson is to demonstrate that such a challenge for building an autonomous ground vehicle (AGV) to navigate in urban environments can be rapidly and economically addressed via use of a general purpose programmable robotics software platform. The team's results demonstrate that such a general purpose robotics software platform enables the rapid after-market drop-in and integration of commercial off-the-shelf (COTS) sensors, actuators, computing hardware, and software; and how generically defined configuration parameters are modified, and new rules of behavior are rapidly defined to provide new behaviors for autonomous vehicle navigation. The team demonstrates that making the leap from a desert off-road AGV to a city driving AGV is dramatically simplified using a general purpose software platform that facilities an after-market drop-in of COTS technology, rapid codification of new rules, and low-cost fielding of the resultant AGV capabilities. The team's approach may be summarized as follows: 1) *drop-in* general purpose robotics software platform and drivers, 2) *tune* programmable configuration parameters for a specific vehicle, 3) *rapidly add new rules* to the programmable engine for the terrain and type of mission, 4) *test and observe* results, 5) *rapidly re-adjust* configuration parameters and rules to optimize results.

Though Team Jefferson has leveraged and has built upon a significant amount of existing reusable software and hardware technology that was used in their 2005 DARPA Grand Challenge semi-finalist vehicle 'Tommy', we recognize that the Urban Challenge presents a radically different set of problems for autonomous vehicles to address. For instance, navigating in an urban environment requires the ability to provide vehicle separation, staying in lane, road following, passing, following traffic rules (*e.g.*, stopping at intersections), dynamic route re-planning, merging, and traffic responsiveness. In spite of the different problem sets, our approach has not been to re-build Tommy from scratch using one technology for the desert and one for the urban environment. Rather, Perrone Robotics' MAX robotics software platform on which Tommy is based was designed specifically to handle radically different environments and applications. As a result, Tommy Jr., is simply a new instance of Tommy with a different vehicular body and rule set specifically geared for the urban environment. We have been able to re-use much of the software already developed and have focused on two major areas: (1) interface with a new reliable vehicle platform, and (2) development of the "driving rules" for urban environment navigation.

3.0 System Architecture

Figure 1 shows the hardware architecture and Figure 2 shows the software architecture for Tommy Jr. Section 4.0 provides the narrative for the system components in the system architecture depicted in Figure 1. This architecture leverages many of the same components implemented in Tommy. In other words, many similar design, hardware and software components, are used to address the desert and urban problems as different as they are. Team Jefferson thus started out with a significant baseline of technology in autonomous ground vehicles that has also been commercialized by Perrone Robotics.

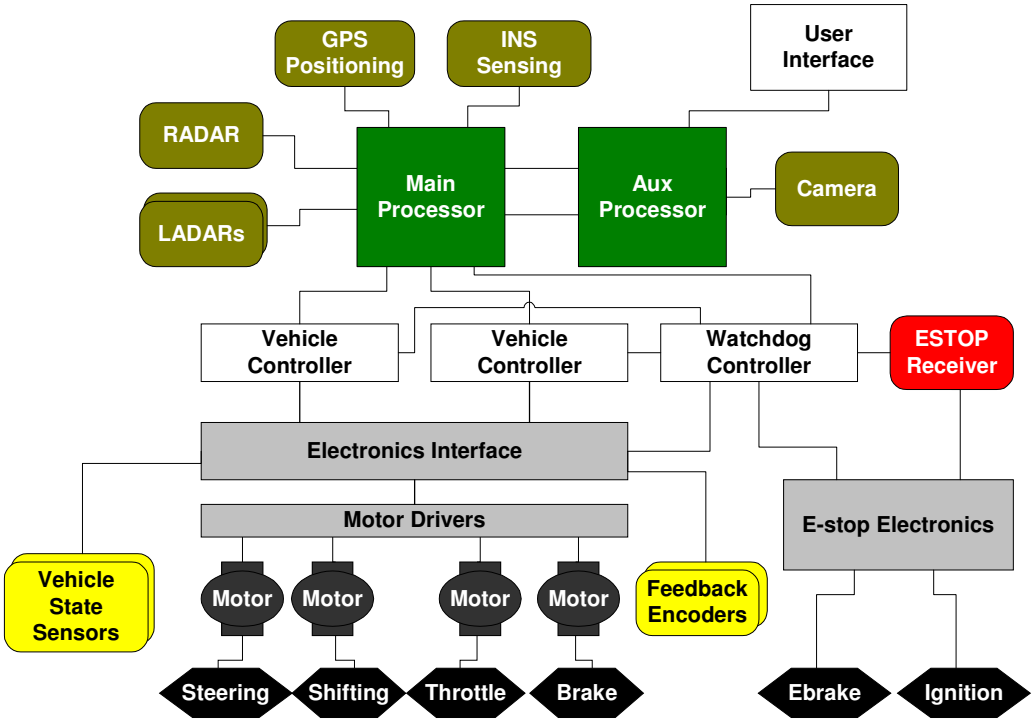

Figure 1: Hardware architecture for Tommy Jr.

Figure 2 shows the software architecture for vehicular control of Tommy Jr. The "brains" of the navigation and traffic control algorithms are run in standard processor environments, which include a COTS processor, operating system, and the Java Real Time System (Java RTS) virtual machine. The MAX software platform provides the core robotics processing environment [3, 4]. The MAX-UGV framework provides core functions for navigation and autonomous vehicle control. A MAX-Rules framework is used for codification of rules. A suite of sensor and actuation drivers for specific sensors and controls are also provided in this environment. Application specific logic and rules for urban challenge behaviors are captured in the Urban Challenge Application Customization/Rules layer (Figure 2).

The core MAX Common engine for sense-plan-act functionality is used in the embedded controller environment as well. The controllers in Tommy Jr. leverage this core MAX engine in a Micro profile with modules for implementing open and closed loop feedback control of motors/actuators using encoders as well as for filtering of analog and digital I/O. The operational integration of the system components works as follows. Within the MAX Standard profile running on the main processor, the sensory data is received and fused, a series of rules are applied to determine what plan of action should be undertaken, and then commands for actuation are issued to the controllers. The controllers then go through the same basic sense-plan-act approach using the MAX Micro Profile in the embedded control environment. The MAX Micro profile receives commands from the main processor to actuate the drive-by-wire motors to a particular position. The command information is treated as an external sensor that directs the micro-controller. The planning components of PRI-MAX Micro on the micro-controller then employ standard feedback control algorithms to direct the motors to their desired position using information fed back via encoders and vehicle state sensors.

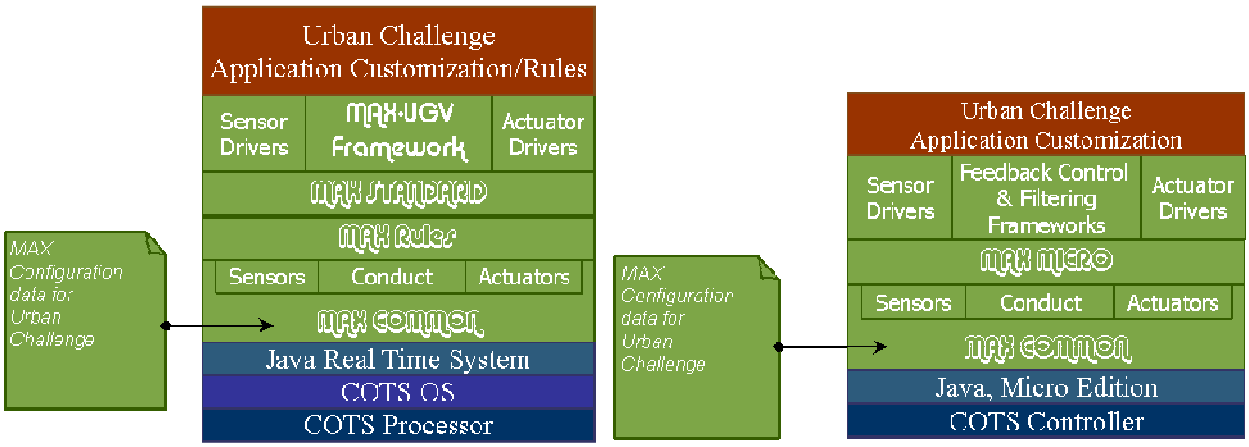

Figure 2: Software architecture for processors (left) and controllers (right)

Whereas a typical robotic application development process would require extensive customization and software components built from the ground up for each desired function, the unique method employed by Team Jefferson is to use Perrone Robotics' MAX platform to facilitate a rapid and cost effective application development approach. For example, this architecture was implemented in the form of Tommy, a robotic dune buggy, for the 2005 DARPA Grand Challenge (Figure 3), then was rapidly re-hosted inside of a low-cost robotic go-kart platform (Figure 4). The same MAX platform technology also continues to be rapidly extended to various commercial robotics and automation applications, e.g., unmanned air vehicles, autonomous tractors, bulk goods measurement, and industrial robotics [5].

Figure 3: Tommy UGV

Figure 4: KartBot UGV

While the same basic hardware/software framework is re-used for Tommy Jr., a robust and reliable new realization of an autonomous vehicle for safe urban zone operation now includes: 1) a new safe and reliable on-road vehicle with a full-size stock chassis based upon a Scion xB automobile shown in Figure 5; 2) actuators, electronics, and power systems that are reliable with noise immunity and power regulation carefully considered and designed into the system by professionals; 3) reliable controllers hosting our software for feedback control of all actuation points; 4) processor platforms with a real time and deterministic processing environment; 5) a redundant watchdog controller for the detection of hazardous conditions and fail-safety of the vehicle; 6) reliable discrete components for fail-safety of the vehicle based on watchdog controller and E-stop system signals. Figure 1 shows the architectural relationship for these

components in Tommy, Jr.

4.0 System Components

In this section we provide an overview of the various components that make up the Tommy Jr. AGV by type: automotive plant, mechanical, electrical/electronic, sensors, computing hardware, and software engine.

Figure 5: Tommy Jr. UGV

4.1 Automotive Plant

The automotive plant includes the chassis, engine, shell, and the car platform in general. A Scion xB (Figure 5) with automatic transmission is used as the full-size stock chassis and basis upon which the automotive solution is built. The Scion xB was chosen as an economical platform known for its maneuverability and agile handling in city driving conditions. The Scion xB also has a modular exterior design facilitating mounting of sensors, and a spacious modular interior facilitating housing and mounting of equipment. The weight of the vehicle is approximately 2425 lbs, has a wheelbase of 98 inches, and measures approximately 155 inches long x 66 inches wide x 64 inches tall.

4.2 Mechanical Components

The mechanical components are those mechanical parts that provide the linkage between the automotive plant and the components used for autonomous controls. All mechanical components including actuators for vehicle drive points (brake, throttle, steering, shift, E-brake) as shown in Figure 1, sensor mounts, platforms, housings, damping controls, and a high output 12V alternator are COTS components dropped into the existing automotive plant.

4.3 Electrical/Electronic Components

The electrical and electronic components are those parts that power the system and provide the bridge between the computing equipment and mechanical components. An array of batteries are used to power the vehicle including the automotive plant itself, actuators, sensors, electronics, and computing equipment. Power filters, motor drivers, and noise shielding are used to provide clean power sources to the onboard equipment and provide isolation from automotive and actuator power draw. Relay banks are used to provide on-demand swapping between human control and computer control of actuators as well as enable the emergency E-stop kill circuitry. A few custom electronic circuit and switch boards interface between micro-controllers and low-level sensors, encoders, and motor drivers (illustrated In Figure 1).

A custom circuit board for cutting power to the engine and triggering the e-brake is used for the E-Stop system (Figure 1). A wireless transmitter and receiver are used to remotely trigger the E-stop system and is designed to provide a modular interface that allows a DARPA supplied E-stop system to be easily dropped in as a replacement. Fail-safe and highly reliable circuit designs employ reliable discrete components, watchdog circuits that expect a periodic pulses from the micro-controller arrangement, parallel arrangement of fail-safe signals from E-stop buttons and the E-stop system, and a fail-safe design such that if a periodic pulse signal is lost, the circuit will fail safe and trigger the engine cut-off and e-brake.

4.4 Sensor Components

The sensor components are those components that are used to sense the vehicle environment and state and include a global positioning system (GPS), inertial navigation system (INS), Laser radars, RADAR unit, stereo vision camera, encoders, and vehicle state sensors as shown in Figure 1.

CSI-Wireless' DGPS MAX system is used as the primary GPS positioning signal source. The DGPS MAX unit is configured to provide OmniSTAR corrected GPS information with sub-meter accuracy. The unit also employs CSI Wireless' COAST technology allowing the unit to provide accurate positioning information for up to 30 minutes during signal outages. As a diversely redundant positioning source, a CSI Wireless Vector Sensor PRO unit provides differentially corrected GPS information from beacon-based DGPS signals.

GPS outages are handled in a series of gracefully degraded states. The DGPS MAX unit provides the highest level of positioning accuracy from OmniSTAR corrected signals. As GPS errors are introduced and outages occur, the graceful degradation plan leverages output from CSI Wireless COAST technology and beacon-based DGPS information from the Vector Sensor PRO. As the confidence in positioning information degrades, the speed of the vehicle is regulated to safer travel speeds. In the event of a complete outage, a minimum speed is utilized and the

vehicle's range sensors are leveraged along with dead reckoning until accurate GPS information becomes available.

The main function of the Vector Sensor PRO is to provide orientation information. A combination of moving base station Real Time Kinematic (RTK), gyro, magnetic compass, and tilt sensor (accelerometer) are used to provide accurate heading and pitch information with 0.1 degrees accuracy. Roll information is yielded directly from the tilt sensor.

SICK LMS laser radars (LADARs) are used to provide range information that is used to deduce positive, negative, and surface obstacle information. Lasers are mounted on the front, the sides, and rear of the vehicle for precise obstacle detection and localization relative to the vehicle. Range information of up to 80 meters is provided and used to determine if there are obstacles in the vicinity of the vehicle. The maximum field of view possible for each LADAR is 180 degrees.

Additionally, an Eaton Vorad RADAR unit with 150 meter maximum range and 12 degree field of view is also mounted on the front of the vehicle. The RADAR provides forward looking obstacle detection at a farther range but with less resolution than the LADARs. As such, it primarily is useful for longer range obstacle warning as opposed to precise location identification sensor. It is also useful as an obstacle detection backup in those scenarios of low reflectivity that affect the performance of the LADARs.

Point Grey Research's Bumblebee 2 stereo vision camera is also used on Tommy Jr. The forward looking stereo vision camera is used as an additional source of obstacle detection and recognition. The camera is also used in the detection of lane and road boundaries.

Encoders for detecting actuator position are used for feedback control of actuators. Sensors to detect the state of the vehicle such as speedometer, tachometer, and throttle position are integrated and used for speed governing.

4.5 Computing Hardware

Java technology-based micro-controllers are used as the micro-control platform for feedback control of the vehicle's actuators integrated with feedback encoders and vehicle state sensors as illustrated in Figure 1. Two vehicle controllers are used to control the vehicle's steering, shifting, throttle, and brake controls. Commands (e.g. desired steering angle, desired speed) from the main processor are sent to the micro-controllers over a serial port.

A dedicated micro-controller is used to implement a simplified safety watchdog monitor to collect and evaluate information from the main processor and other micro-controllers. The watchdog controller sends a periodic safety signal to the E-stop electronics board. Absence of the watchdog signal to the E-stop board triggers fail-safe operation.

Low-power x86 processor cards are used as the onboard processors. A main processor receives GPS, INS, LADAR, and RADAR sensory information, and performs all navigation, sensor fusion, obstacle avoidance, route planning, and traffic rules following functionality. A second processor performs all processing for the vehicle's stereo vision camera and also provides

a graphical user interface for configuring, testing, and operating the autonomous vehicle. The two processors are connected by a high-speed local area network and the main processor handles all direct communications with the controllers over serial port connections.

4.6 Software Platform

The software architecture (Figure 2) consists of a standard processing environment and the embedded controller environment. Sun Microsystems' Java Real Time System (RTS) runs on the standard processing environment and provides a real-time deterministic processing environment. In the embedded controller environment, an embedded Java Micro Edition (ME) runtime environment is employed as the standard embedded controls platform.

Perrone Robotics' MAX-Common engine features a common sense-plan-act framework and libraries for robotics that run in both standard processor and embedded controller environments. Perrone Robotics' MAX-Standard profile is used to provide standard processor robotics controls and is used with a MAX-RealTime plug-in on the standard processors to provide real-time deterministic controls. Perrone Robotics' MAX-Micro profile with drivers for the Java-based micro-controller hardware is used to provide micro-level robotics controls. Existing software drivers for the various sensors and actuation approaches are used, including drivers for SICK lasers, Eaton Vorad RADARs, GPS, INS, feedback encoders, pulse-width-modulated motor controls, and drivers for speedometer, pulse-counting, analog I/O, digital I/O, and RPM sensing. A feedback control framework built atop MAX-Micro is used for the open and closed loop PID-based feedback control. The MAX-UGV framework is used for providing common navigation and obstacle avoidance services on the standard processors.

The robot's decision making is governed and refined by a simplified planning and rule engine framework built into MAX. Existing rules for autonomous navigation, obstacle detection, and obstacle avoidance behavior are currently implemented inside of MAX-UGV software components. An initial rule set and components that encapsulate urban driving behaviors have also been implemented inside of the rule engine. The team thus had access to a foundational library of rules that synthesize decisions and behaviors governed by route planning constraints, traffic rules, road following constraints, obstacle detection rules, and obstacle avoidance rules. New rules based on experiential urban zone driving have been rapidly codified inside such a planning and rules framework.

5.0 Analysis and Design

This section describes the design choices made for individual features of the overall system to address the problem at hand. Preliminary results from these design choices are also presented.

5.1 Steering Control System

Given a desired steering angle, a mechanism for precisely positioning the steering column of the vehicle is required. Steering controls are achieved at two levels of abstraction. On the main processor, desired steering angles for the vehicle are assessed as a function of the desired steering navigation rules and by using high-level Proportional-Integral-Derivative (PID) controls [6]. At the micro-controller level, the steering is achieved via low-level PID positioning controls according to a standard control equation:

Error = DesiredValue - CurrentValue

ControlValue = PID(Error)
= Kprop x Error + Kderiv x Derivative(Error) + Kinteg x Integral(Error)

where, the *DesiredValue* is a desired position, *CurrentValue* is a current position, *Kprop*, *Kderiv*, and *Kinteg* are constants, *Derivative()* is the derivative of the error over time, *Integral()* is the integral of the error over time, and the *ControlValue* is the resultant control output.

Software components available in the MAX framework provide a configurable means for implementing PID controls. No programming is necessary using such components. Rather, all PID behavior is defined via configuration parameters. Additionally, a MAX software motor driver and MAX software feedback encoder sensor driver are configured and registered with the MAX feedback control component running within the MAX-Micro process on a Java-based vehicle controller. These available components provided by the MAX framework are configured simply by specifying the relevant control parameters for steering feedback control such as a few that are shown here:

Kprop = constant for proportional controls
KDeriv = constant for derivative controls
Kinteg = constant for integral controls (0 used here for Tommy Jr.)
MaxValue = maximum allowed output value
Resolution = resolution for achieving position
MotorDriver = type of generic driver to load for controlling the motor (e.g. DCServo)
FeedbackDriver = type of generic driver to load for feedback (e.g. QuadEncoder)
UnitChangePerFeedbackUpdate = encoded units to change feedback position based on a feedback sensor update event

Thus, in addition to specifying the appropriate software driver types to load for the particular motor type being controlled (e.g. a DC servo motor driver) and feedback sensor type employed (e.g. a quadrature encoder), all that is required are to configure a number of control variables such as the ones defined above to rapidly activate feedback controls for steering. For steering control, given a commanded steering angle from the main processor, the MAX feedback control components then transparently handle PID controls in real-time for the steering motor such as obtaining the steering error, deriving motor control direction, checking for max positions and stall conditions, checking for achieved position resolution, and actuating the motor in the proper direction and speed. Thus, for steering control we have the following behavior provided for us transparently by the MAX engine:

SteeringError = DesiredSteeringPosition - CurrentSteeringPosition
SteeringControlPosition = PID(SteeringError)

While simulations are possible for identifying the appropriate selection of configurable control parameters, because the means for configuring such variables is easily performed, a more direct means of dynamically adjusting parameters during initial test runs was performed. Within a matter of minutes, reasonable PID constants, resolutions, and other control parameters were identified while running the vehicle live under MAX feedback control. The results come in the form of directly experienced steering controls of the vehicle live. Swerving and over-shooting of position are quickly addressed during parameter tuning. Further tuning is observed by examination of off track distance values generated when commanding the vehicle to navigate

straight and curved paths. As the image depicted in Figure 6 illustrates, an initially charted course (shown in blue) was first used to create a few straight runs and turns. Using freshly deployed default and guessed values for steering constants, the red route shows the somewhat uncontrolled turns as a result. Examination of logs demonstrated that the proportional constants were producing too much fluctuation. A dynamically adjusted set of constants were created and the vehicle re-run yielding the much smoother run as depicted by the green route. Tommy Jr's preliminary steering control parameters were in fact tuned over the course of a few hours total ultimately yielding off track distance errors of under 12 inches.

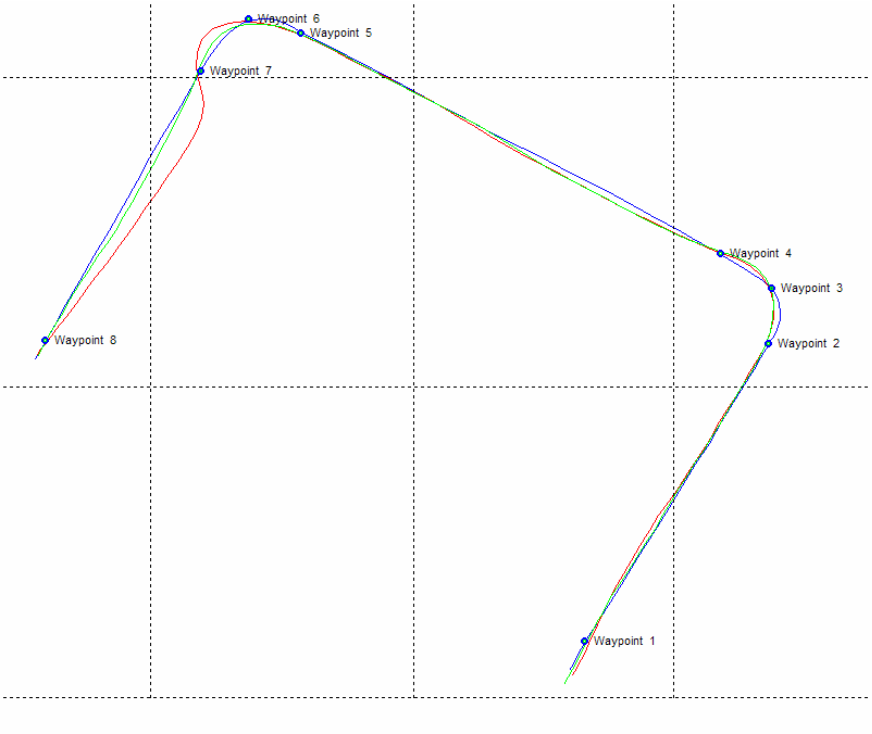

Figure 6: Rapid Steering Tuning

5.2 Speed Control System

Speed controls for Tommy Jr. are also achieved at two levels: high-level controls on the main processor and low-level feedback control on the vehicle controller. For control of both the throttle and brake, the same MAX feedback control framework components used for steering position control are leveraged for PID control of throttle position and brake position. While different feedback sensors and motors are used for throttle and brake, MAX drivers provide an abstraction layer for such sensors and motors, and enable the same MAX feedback control approach and configuration parameters described for steering control above to be employed.

Additionally, an intermediate level of control is used to translate desired speed values commanded from the main processor into desired throttle position and brake position values. For brake level, another MAX feedback control component is configured to read the current speed from a speed sensor and compute the error differenced with the commanded desired speed from the main processor. The desired brake position is then computed as a linear function of the PID control output of the speed error. The desired throttle position is also computed as a function of

the speed error and added to the current throttle position. Both desired throttle and brake positions are then fed into the individual MAX feedback control components for throttle position and brake position, respectively. Thus, we have the following behavior transparently handled for us by the MAX engine with just a simple matter of setting configuration parameters:

SpeedError = DesiredSpeed - CurrentSpeed
DesiredBrakePosition = LinearFunction(PID(SpeedError))
DesiredThrottlePosition = CurrentThrottlePosition + PID(SpeedError)

BrakeError = DesiredBrakePosition - CurrentBrakePosition
BrakeControlPosition = PID(BrakeError)

ThrottleError = DesiredThrottlePosition - CurrentThrottlePosition
ThrottleControlPosition = PID(ThrottleError)

Thus, the highly configurable MAX feedback control components also dramatically reduced the time for implementing speed controls for Tommy Jr. Simply by specifying the drivers to be loaded and their configuration parameters, Tommy Jr's speed controls were ready for tuning within minutes of configuration. By riding in the vehicle and tuning configuration parameters dynamically, within a few hours time, acceptable initial speed controls were reliably regulating the new Tommy Jr vehicle platform speed to within 2 miles per hour.

5.3 Steering Planning and Rules

While the steering controls described in Section 5.1 apply to low-level embedded controls of a steering servo motor, a higher-level of control is required to deduce a desired steering angle and refine the steering controls based on vehicle dynamic information only available on the higher-level main processor. Steering components that form part of the MAX UGV framework are leveraged for implementation of Tommy Jr's high-level steering controls. On a synchronous basis, configured to be 100 milliseconds, a MAX UGV rule is invoked to compute a target steering waypoint. First a target waypoint is computed and projected along the currently planned route for the vehicle as a function of the vehicle's speed, current position, current route, and allowable course boundaries. The desired course to that projected position is then computed and used to derive a steering error as illustrated here:

MovingWaypoint = WaypointProjection(CurrentSpeed, CurrentPosition, CurrentRoute)
DesiredCourse = CourseProjection(MovingWaypoint)

A sequence of conditions are then evaluated to determine the appropriate MAX feedback control PID parameters to leverage on the main processor. Depending on conditions such as whether or not the vehicle is simply navigating a course with no obstacles in sight, making a sharp turn on a narrow pass, navigating around obstacles, is in a U-turn state, or other programmable conditions, particular PID parameters are used to produce the desired steering angle. This desired steering angle is then used to command the vehicle controller for steering. The steering behavior is illustrated here:

SteeringError = CurrentHeading – DesiredCourse
PID = PIDSelection(vehicle conditions)
DesiredSteeringAngle = PID(SteeringError)

This highly configurable and programmable MAX rules-based approach for defining how to steer a vehicle uses many of the existing rules defined for the offroad desert driving Tommy AGV, and has benefited from new rules for urban driving conditions such as navigating U-turns. For example, it was discovered during testing that more aggressively defined PID parameters are more appropriate for U-turn maneuvers. While such parameters may be precisely mathematically defined, in practice, no one control model is appropriate for all scenarios and for all vehicle types given various vehicle dynamics models. This highly and easily configurable approach for specifying steering rules and behavior based on navigation information has provided a rapid way for quickly evolving Tommy Jr for urban driving behaviors.

5.4 Speed Planning and Rules

The current desired speed for Tommy Jr. is computed as the least restrictive speed allowable based on a series of discrete rules for speed control. That is:

DesiredSpeed = LeastRestrictiveSpeed(rule1, rule 2, …)

The various configurable and reusable speed control rules that have been implemented to date are listed here:

AbsoluteMaxSpeed = max speed allowable for vehicle
TrackSpeed = max speed allowable for the current vehicle track segment
RouteSpeed = max speed set for the current dynamically planned route segment
VehicleSpeed = max speed set for current vehicle state as function of orientation
PitchSpeed = max speed allowed as function of filtered vehicle pitch
ReactionSpeed = max speed set for reacting to obstacles ahead
NarrowPassSpeed = max speed allowable as function of passage size
TurnSpeed = max speed allowable as function of upcoming turns
ProgressiveAggressiveSpeed = max speed allowed as function of an acceleration filter

Thus a series of high level rules are defined based on a wide variety of vehicle conditions that define a maximum allowable speed based on those conditions. The rules to use for the particular vehicle are specified in a configuration source and dynamically loaded and registered with the MAX rule engine. At runtime, upon rule execution, the least restrictive speed is selected and delivered as a commanded speed to the vehicle controller.

New rules are easily added as new rules of behavior are defined. Existing rules are easily configured based on a variety of configuration parameters for each rule. For example, the progressive aggressive speed control filter is based on the principle that when a restrictive speed condition is identified (e.g. obstacle detected), and if that speed condition is removed (e.g. obstacle disappears), then the speed of the vehicle should only conservatively increase over time by some small increment as it becomes certain that the obstacle has disappeared from the field of view. The progressive aggressive speed rule has a configurable *SpeedIncrementPerUnitTime* parameter. For the Grand Challenge driving scenario, where detected obstacles were likely to be static, this parameter was set to be relatively low. For the Urban Challenge scenario where detected obstacles may also be moving, this parameter is set to be higher. Such tuning of existing rules and easy plug in of new rules using the robust MAX UGV framework have dramatically simplified the evolution and tuning of Tommy Jr's behavior.

5.5 Navigation Planning and Rules

A series of high-level navigation planning components and rules have been defined as part of the MAX UGV framework. Tommy Jr has leveraged these abstractions to dynamically plan a route based on route information and on dynamic obstacle information. An A* algorithm is used to traverse a route network generated from a DARPA-specified RNDF file and identify an optimal route along the network given a series of checkpoints generated from an MDF file [7, 8]. This route is then used to define the optimal course through which the vehicle may travel in completion of its mission. High-level components built into the MAX UGV framework have simplified the construction of such routes from route information and mission information defined independent of the actual physical format by which such information is loaded into the system. The route planning components themselves are also independent of the underlying optimization algorithm selected. At the time of this paper's creation, the team has only leveraged the A* algorithm in formulating a route plan based on an RNDF-defined route network and MDF-defined mission of checkpoints as illustrated here:

LongRangeRoutePlanning
RouteNetwork = LoadMap(RNDF)
Mission = LoadMission(MDF)
Course = RoutePlanning(RouteNetwork, Mission)

Given a long range planned course, Tommy Jr begins to identify current target waypoints based on the traversal of such a course network given its current position localized to that course as a GPS position. As waypoints are reached on the course, Tommy Jr traverses to the next waypoint. A moving target waypoint, as described in Section 5.3, defined along the course of the vehicle is used as the waypoint for projection of desired steering. The dynamic route of the vehicle may be altered by obstacle planning and rules as well as by navigation and traffic rules.

As obstacles on the course are identified, valid 'openings' on the course ahead through which the vehicle may travel are identified based on track boundaries and detected obstacle dimensions. The rules used to implement the decision making for determining what are valid openings has been defined completely parametrically within the MAX UGV framework. Hence new behaviors for urban driving, such as defining a valid opening width on a lane, simply required that such configuration parameters be tuned.

After a sequence of openings on the course ahead are identified, a series of short range route planning rules are executed to identify the optimal route. While the same generic route planning infrastructure for the MAX UGV framework is leveraged for short range dynamic route planning based on detected obstacles, a different underlying evaluation and cost function is configured for use in generating the dynamically planned short range route. The generated short range route is based on minimizing the number of turns the vehicle needs to make in avoiding obstacles. Such a cost function is leveraged for short term route planning, whereas a cost-function using the A* algorithm for long range route planning minimizes the distance for the vehicle to travel.

Upon generation of a short range dynamically planned route, the route is queried for any identified exception scenarios along the way. The route planner consults a configurable array of registered rules to produce such exceptions. These rule evaluation functions include the

identification of road block impasses, impasses that may be passed in an alternate lane, u-turns, etc. The overall navigation planning process is illustrated here:

ShortRangeRoutePlanning
Openings = OpeningIdentification(ResolvedObstacles, Course)
Route = RoutePlanning(Course, Openings)
Maneuvers = HandleExceptions(Route)

Tommy Jr leverages the same MAX UGV framework components for short range route planning as the desert driving vehicle Tommy. New rules executed during route planning have been added for detecting the need for passing vehicles, rerouting based on road blocks, U-turns, and other conditions encountered in an urban driving setting. Additionally, while new components have been added to the MAX UGV framework for loading route networks and missions, the team was able to leverage the same route planning framework code, albeit configured for different cost functions and optimization algorithms, for Tommy Jr. Preliminary results have demonstrated that the A* algorithm is sufficient for long range planning of the vehicle's course and for short range planning of the vehicle's dynamic route around obstacles. While offline tests have yielded favorable results for longer range route planning, the ease with which new cost functions and optimization algorithms can be added may be leveraged to further optimize Tommy Jr's ability to navigate larger route networks.

5.6 Obstacle Planning and Rules

Tommy Jr. leverages a generic and configurable process within the MAX UGV framework for dealing with obstacles. Data from sensors used to identify obstacles is either asynchronously or synchronously received by the MAX engine. Generic MAX UGV framework components and rules for detected obstacles based on sensory data are then invoked with sensor specific rules and concrete realizations of generic components. Generic obstacles defined based on such rules are then created dynamically and passed to a generic obstacle resolution component. This component acts as a sensor fusion module for fusing the data from various sensors to assign a 'level of confidence' to different identified obstacles. The obstacle resolution components also serve as 'obstacle memory' recalling whether or not obstacles have been seen before, regardless of whether or not the obstacle was detected previously by the same sensor or by different sensors. After the detected obstacles are resolved, the resolved obstacles with properly assigned confidence levels and characteristics are passed onto the short range planning components as described in Section 5.5 for use in defining dynamic routes. The basic process for obstacle handling is defined here:

SensorData = SensorInput()
DetectedObstacles = DetectObstacles(SensorData)
ResolvedObstacles = ResolveObstacles(DetectedObstacles)
ShortRangeRoutePlanning() … as described in Section 5.5

For obstacle detection, specific components and rules associated with the data coming from particular sensors are used to scan sensor data and distinguish potential obstacles. Different concrete rules for such data are triggered in formulating decisions and characteristics such as obstacle type (e.g. surface, positive obstacle, negative obstacle), obstacle dimensions, obstacle location, and a confidence level. The rules for such decisions vary from sensor to sensor. For example, a higher confidence level for a detected obstacle based on laser data may decrease as

the distance from the laser increases due to the loss in resolution of laser data. While such rules vary from sensor to sensor, the output of the detection process are collections of detected obstacles described in a generic fashion.

The general purpose obstacle resolver takes this collection of generically defined obstacles and goes through a process of comparison with existing obstacles in memory. If a recently detected obstacle is similar to an obstacle in memory, the obstacle characteristics are merged with newly detected obstacle data superceding older data. The confidence level of the merged obstacle may also be increased if the obstacle has been repeatedly detected. Likewise, the confidence level of obstacles in memory are also decreased over time as they are repeatedly not re-detected. Finally, the resolver removes obstacles from memory that are no longer valid or of a confidence level that warrants their recollection. Once again, the parameters for defining how obstacles are recalled and evaluated by the MAX UGV resolver are completely configurable with new rules for evaluation that are easily added. Tommy Jr has leveraged such configurability for urban driving conditions whereby obstacles may be moving. For example, new rules for increasing the confidence of an obstacle that has moved along an identifiable path have been added to deal with common urban driving obstacles such as moving vehicles.

During the short range planning process, resolved obstacles are queried for their dimension and location information when calculating passable openings on the route. A configurable engine and set of rules are defined which evaluate obstacle confidence levels and other characteristics such as distance from vehicle and distance outside field of view to derive two additional characteristics for route planning. The first derived characteristic is whether or not an obstacle should be used for avoidance. That is, is it of a high enough confidence and relevance to be used in the creation of openings through which the vehicle may avoid the obstacle? The second derived characteristic is the speed at which the vehicle should travel in the obstacle's presence. Thus, while a distant obstacle may not be avoided immediately, it may induce the vehicle to slow down to a speed such that it could avoid it if the vehicle gets close enough to the obstacle. These rules may be summarized as follows:

Avoid = ShouldObstacleBeAvoided(confidence, distance, field-of-view-distance)
ReactionSpeed = ReactionSpeed(confidence, distance, field-of-view-distance)

5.7 Maneuver Handling Planning and Rules

Thus far, the team's configurable rules-based approach for controlling the vehicle's speed and direction, regulating the vehicle speed, navigation, and obstacle handling have been described. Many of the Urban Challenge's navigation and traffic, both basic and advanced, required behaviors are addressed within such a framework. For example, maintaining a safe vehicle separation distance is achievable by configuring parameters associated with the calculation of vehicle reaction speeds. While such speeds are initially calculated based on reaction time, safe stopping distances, and the vehicle dynamics, the ultimate codification of such values are defined simply within a set of configurable rules used in calculating reaction speed tables. Thus different vehicles, different terrains, and different sensors may easily and rapidly be adapted.

However, additional maneuvers and scenarios must be explicitly addressed with concrete rules plugged into such a framework. For example passing a stopped vehicle needs explicit rules

for vehicle behavior modification. When an impasse ahead of the vehicle is detected during route planning, a wait and pass maneuver is registered to be considered on a synchronous basis. As the passing maneuver goes through its rules state diagram (i.e. from stop vehicle, to wait period, to begin passing, to end passing), a synchronous loop continuously invokes the maneuver registered and scheduled to be evaluated. A passing maneuver condition induces this maneuver rule to be evaluated. If the vehicle no longer needs to pass a vehicle ahead (e.g. the vehicle ahead begins moving), then the passing maneuver condition will become invalid, and the vehicle will remove the passing maneuver from consideration. The basic process for evaluating dynamically registered maneuver rules is illustrated here:

```
SynchronousLoop
    For each maneuver to be considered
        If(maneuver trigger conditions met)
            TriggerManeuver()
        If(maneuver removal conditions met)
            RemoveManuever()
```

This simplified rule engine structure within MAX has proved to provide a simplified way to rapidly add new rules of behavior required by the DARPA Urban Challenge. Triggering of maneuver rules ultimately result in adjusted speeds and dynamic routes that are independently evaluated by the existing components of the MAX UGV framework. Because maneuver rules are expressed in a common language using high-level components already part of the MAX engine and MAX UGV framework, the ease by which new rules are added is greatly simplified. What's more, because all rules are defined according to a format pluggable inside the MAX engine, the ability to dynamically load new rules either locally or over a distributed network is possible out of the box. Thus MAX processes accessible on the same network can register, exchange, and modify rules of behavior on the fly.

6.0 Results and Performance

The most significant result from the team's approach has been the verification that rapid and cost-effective construction of an autonomous ground vehicle for the DARPA Urban Challenge is possible by leveraging MAX and its extensions as a highly configurable and programmable robotics software platform. What's more, the after market drop-in approach for building an AGV on a new vehicular platform was made possible with minimal configuration time (i.e. measured in hours). The extensibility of such a platform has also provided a rapid and cost-effective means for adding new rules of behavior and maneuvers required for autonomous navigation of an urban setting. Hence the primary demonstrable result of the team's endeavors has been the development of an AGV that required little time and little cost for realizing basic navigation and traffic functionality. At the time of this paper's writing, the following metrics were gathered summarizing the team's efforts in this regards:

Man-hours for basic AGV configuration: 24
Man-months for basic navigation and traffic rules: 2
Man-months for mechanical and electrical component drop-in: 4
Processing power: 2 low-cost processing cards
Total cost of components: $50,000

The net result of this effort has yielded an AGV that is rapidly evolving to satisfy all of the

technical evaluation criteria to be considered for the DARPA Urban Challenge. While a more precisely quantified approach for the results of the vehicle's various rules performance may be ultimately desired, the team has employed a more economical and direct evaluation approach of *configure-test-observe-analyze-refine*. That is, newly *configured* behaviors are immediately *tested*. Vehicle behavior is then empirically *observed*. Upon any anomalous or undesirable behavior during testing, logged data from that run is *analyzed* to assess what computations were made and what conditions were evaluated to trigger the specific undesirable behaviors. The specific reason for the undesirable behavior is then precisely identified and a mitigating solution is defined and implemented. Most often, the mitigation plan boils down to a *refinement* of existing rules or of existing parametric variables. Another series of tests are then performed to verify that the undesirable behavior is mitigated. This process is iteratively repeated for each newly deployed behavior. Although not generally and always quantifiable, such an approach to testing new features provides a rapid and direct qualitative approach to identifying anomalies on the actual vehicle under development. This further facilitates a more natural mapping from high-level human understandable problem assessments to rapidly refined high-level rules of behavior codified into the vehicle.

7.0 Conclusions

Team Jefferson's after-market drop-in of COTS hardware rendered Tommy Jr AGV hardware capable in just four man-months of time with a total parts cost of approximately $50,000. In just under one week's time, Tommy Jr's MAX software platform, drivers, and MAX-UGV framework software was installed and configured to yield a fully autonomous vehicle capable of autonomous navigation and obstacle avoidance. An additional two man-months of time were spent defining and dropping in rules for basic navigation and traffic behaviors for the DARPA Urban Challenge setting. Currently, all of Tommy Jr's sensing and decision making logic runs on two low-cost, low-power processor cards costing less than $200 a piece. Team Jefferson has been able to demonstrate using an economical and direct configure-test-observe-analyze-refine approach that navigation and traffic behaviors for an Urban Challenge setting may be rapidly and economically realized. What's more, by using MAX as a highly configurable and programmable general purpose robotics platform along with MAX extensions for UGV operation, the team has demonstrated that a fully capable city driving AGV may be built at a pace and affordable level unparalleled by current state of the art.

8.0 References

[1] *Urban Challenge Technical Evaluation Criteria*; Defense Advanced Research Projects Agency; March 16, 2007; http://www.darpa.mil/grandchallenge/docs/Technical_Evaluation_Criteria_031607.pdf

[2] *Urban Challenge Rules*; Defense Advanced Research Projects Agency; March 15, 2007; http://www.darpa.mil/grandchallenge/docs/Urban_Challenge_Rules_031507.pdf

[3] *MAX Datasheet*; Perrone Robotics; http://www.perronetech.com/public/MAX.pdf

[4] *MAX Programming Guide*; Perrone Robotics; to be published October 1, 2007; www.perronerobotics.com

[5] *All about Java Technology Based Robotics*; Paul J. Perrone; JavaOne 2007; TS-1519; San Francisco, CA; May 8, 2007; http://developers.sun.com/learning/javaoneonline/2007/pdf/TS-1519.pdf

[6] *An Overview of Proportional plus Integral plus Derivative Control and Suggestions for Its Successful Application and Implementation*; David Sellers; http://www.peci.org/library/PECI_ControlOverview1_1002.pdf

[7] *Generalized best-first search strategies and the optimality af A**; Rina Dechter, Judea Pearl; 1985; Journal of the ACM 32 (3); pp. 505 - 536.

[8] *Urban Challenge Route Network Definition File (RNDF) and Mission Data File (MDF) Formats*; Defense Advanced Research Projects Agency; March 14, 2007; http://www.darpa.mil/grandchallenge/docs/RNDF_MDF_Formats_031407.pdf